To estimate the size of the largest two-dimensional array of int values your computer can hold, we need to consider a few factors:

1. Available RAM
2. Size of an int value
3. Operating system and programming language limitations

Let's break this down step-by-step:

1. Determine available RAM: Check how much free RAM your computer has. Let's say you have 8 GB ($8,589,934,592$ bytes) of free RAM.
2. Size of an int value: Typically, an int is 4 bytes (32 bits) on most modern systems.
3. Calculate the maximum number of int values: Maximum int values = Available RAM / Size of int $8,589,934,592 / 4 = 2,147,483,648$ int values
4. Determine the dimensions of the square array: Square root of $2,147,483,648 \approx 46,340$

So, theoretically, you could create a 2D array of approximately $46,340 * 46,340$ int values.

However, this is a theoretical maximum. In practice, you'll need to consider:

- The operating system and other running processes need memory too.
- Programming languages may have their own limitations on array sizes.
- Some memory should be left free for the system to function properly.

A more practical estimate might be to aim for using no more than 50-75% of the available RAM.

To get a more accurate estimate for your specific system, you could:

1. Check your actual available RAM.
2. Verify the size of an int on your system.
3. Research any limitations of your programming language and operating system.
4. Write a program that gradually increases the size of a 2D array until it fails, to find the practical limit.

Here's a JAVA program that does that:

```java
public class MaxArraySizeTest {
    public static void main(String[] args) {
        int size = 1000; // Starting size
        int increment = 1000; // Size increment for each iteration

        while (true) {
```

```
            try {
                // Attempt to create a 2D array
                int[][] array = new int[size][size];

                // If successful, print the current size and free
memory
                System.out.printf("Successfully created array of
size %d x %d%n", size, size);
                System.out.printf("Free memory: %.2f MB%n",
                    Runtime.getRuntime().freeMemory() / (1024.0 *
1024.0));

                // Optional: fill array with a value to ensure
memory is actually allocated
                // fillArray(array);

                // Increment size for next iteration
                size += increment;

                // Clear the reference to allow garbage collection
                array = null;
                System.gc(); // Hint to the JVM to perform garbage
collection

            } catch (OutOfMemoryError e) {
                // If we run out of memory, print the last
successful size
                System.out.printf("Out of memory. Last successful
size: %d x %d%n", size - increment, size - increment);
                break;
            }
        }
    }

    // Optional method to fill the array, ensuring memory is
actually allocated
    private static void fillArray(int[][] array) {
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                array[i][j] = 1;
            }
        }
    }
}
```